Politecnico
di Torino

DBMG

Web Applications

Introduction to Streamlit

Web Applications

- Streamlit
- Text elements
- Input widgets
- Data visualization
- Additional elements
- Layout

Streamlit

Web Applications

Python Libraries

- Python offers several libraries for analyzing, manipulating data, and developing interfaces to facilitate the creation of data analysis applications

Library used to work with datasets. Analyze, clean, explore, and manipulate data

Library (Numeric Python) that allows you to work with numerical data, with multidimensional data structures (i.e., array, matrix)

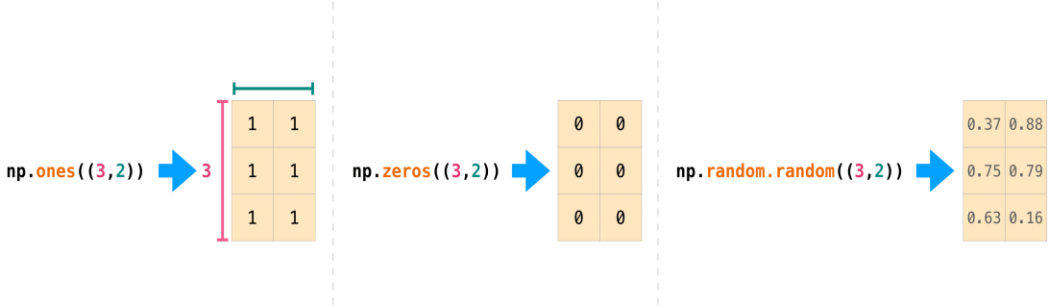
Open-source library that facilitates the creation and development of custom web applications

NumPy

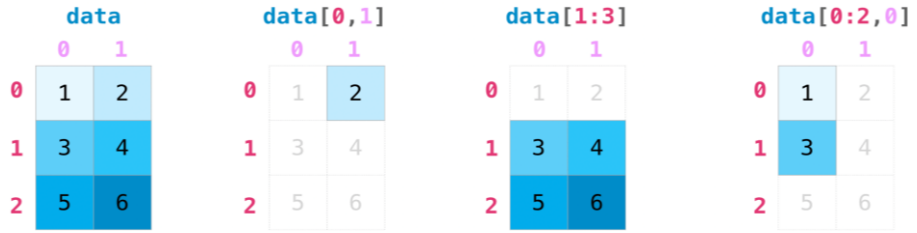
- The main advantages of NumPy are to increase **flexibility** and **efficiency** of operations compared to the native structures of Python
  - import numpy as np
- The data structure revolves around the concept of ***array***, a grid of values referred to as ***ndarray*** (N-dimensional array)
- Dimensions are called ***axes***
- Numpy is the basis of other advanced Python libraries (e.g., Pandas, Scikit-learn)

https://numpy.org/doc/stable/index.html

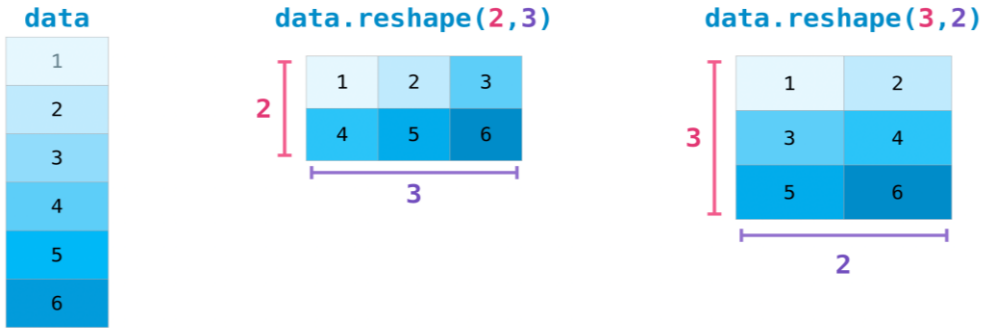
NumPy - Examples



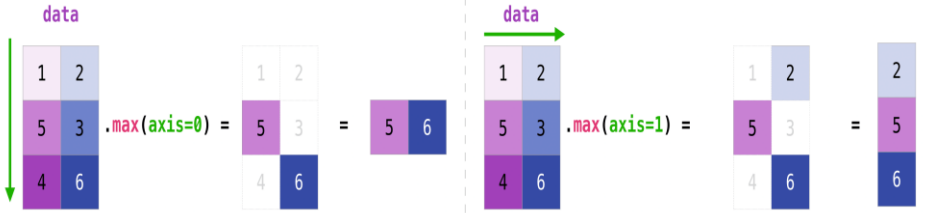
Creation of a matrix



Indexing and slicing



Reshape an array



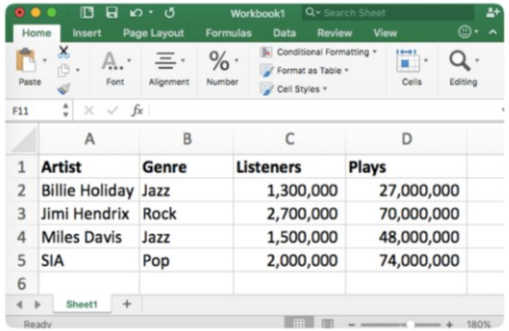
Aggregations on different axes

https://numpy.org/doc/stable/index.html

Pandas

- A fundamental library especially in the field of Data Science together with NumPy (on which it is based)
  - import numpy as np
  - import pandas as pd
- There are two fundamental data structures: **Series** (1-D sequence of homogeneous elements) and **DataFrame** (2-D arrays designed as tables, each column is a Series and has a name)
- Example: browse, analyze, and visualize data from a CSV file

music.csv



	A	B	C	D
1	Artist	Genre	Listeners	Plays
2	Billie Holiday	Jazz	1,300,000	27,000,000
3	Jimi Hendrix	Rock	2,700,000	70,000,000
4	Miles Davis	Jazz	1,500,000	48,000,000
5	SIA	Pop	2,000,000	74,000,000

pandas.read_csv('music.csv')

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

https://pandas.pydata.org/docs/index.html

Pandas

- **Data Loading**: a DataFrame can be created from Series, numpy arrays, dictionaries, JSON, CSV…
- **Data Cleaning**: different functions for data cleaning, removing duplicate data, replacing missing values with default values, converting data types…
- **Data Manipulation**: filtering rows, sorting data, creating new columns, aggregating data…
- **Data Analysis**: view descriptive statistics, create PivotTables, create charts, and other advanced analysis…
- **Data Visualization**: visualization of data with different types of graphs, using the Matplotlib library

Pandas - Examples

```python
import pandas as pd

# Creating a Dataframe from a Dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva', 'Mauro'],
        'Age': [25, 33, 47, 19, 28, 17],
        'City': ['Rome', 'Milan', 'Naples', 'Turin'', 'Florence', 'Turin'']}
df = pd.DataFrame(data)

# Print the first 5 rows of the dataframe
print(df.head())
```

	Name	Age	City
0	Alice	25	Rome
1	Bob	33	Milan
2	Charlie	47	Naples
3	David	19	Turin
4	Eva	28	Florence

```python
# Print column list
print(df.columns)

# Filter rows with age greater than 30 years
df_filtered = df[df['Age'] > 30]
print(df_filtered)
```

```
Index(['Name', 'Age', 'City'], dtype='object')
```

	Name	Age	City
1	Bob	33	Milan
2	Charlie	47	Naples

```python
# Aggregate data by city and calculate the average age
df_aggregate = df.groupby('City')['Age'].mean()
print(df_aggregate)
```

```
City
Florence    28.0
Milan       33.0
Naples      47.0
Rome        25.0
Turin       18.0
Name: Age, dtype: float64
```

```python
# Create a new column that indicates whether the person is
of legal age or underage
df['Adult'] = df['Age'].apply(lambda x: 'Yes' if x >= 18
else 'No')

# Delete the Age column
df_output=df.drop(columns=['Age'])
print(df_output)

# Save dataframe to a CSV file
df_output.to_csv('people.csv', index=False)
```

	Name	City	Adult
0	Alice	Rome	Yes
1	Bob	Milan	Yes
2	Charlie	Naples	Yes
3	David	Turin	Yes
4	Eva	Florence	Yes
5	Mauro	Turin	No

people.csv

Why Streamlit?

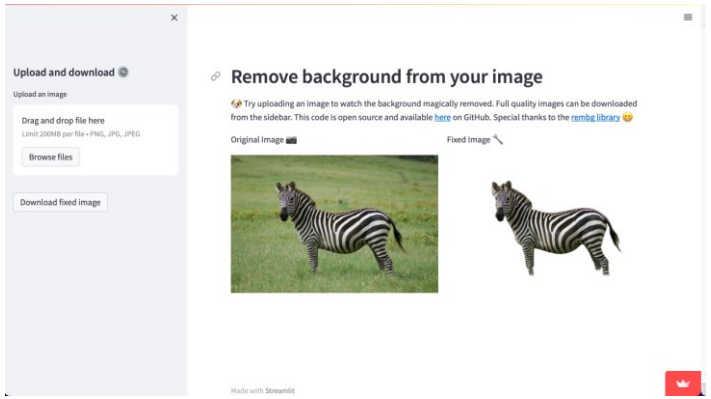
Streamlit

- Open-source Python library that facilitates the creation and development of custom web applications
- Ideal for supporting **data science** and machine learning projects
- You can create interactive interfaces
- Designed for newbies, front-end skills are not expressly required
- Thanks to *widgets* and elements available, you can create web pages with a few lines of code
- Compatible with most Python libraries

https://streamlit.io/

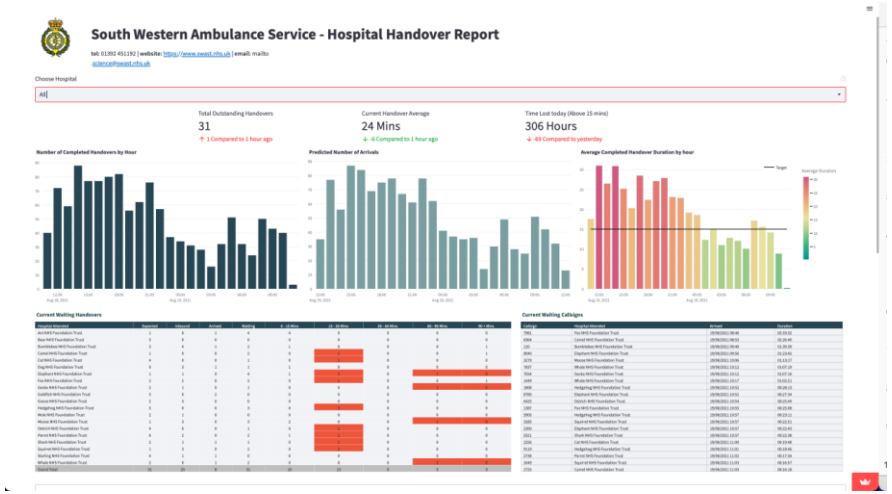
Examples gallery

- There are several templates and applications created by the community (https://streamlit.io/gallery)



Background Removal

Bundesliga analyzer



SWAST - Hospital Handover Report

Installation

- Python 3.7 – Python 3.11
- Using a ***virtual environment*** is always recommended (*pipenv, poetry, venv...*)
- Install Streamlit
  - pip install streamlit
- Test the installation
  - streamlit hello
- Launch your own application
  - streamlit run your_script.py [-- script args]

  Or

  - python -m streamlit run your_script.py

https://docs.streamlit.io/library/get-started/installation

Configuration

- Various possibilities to define configuration options (e.g., server port, theme...) via:
  1. a ***global config file*** (to be created):
     - *~/.streamlit/config.toml* for macOS/Linux
     - *%userprofile%/.streamlit/config.toml* for Windows
  2. *a **per-project*** configuration file:
     - *$CWD/.streamlit/config.toml* where *$CWD* is the folder from which Streamlit was launched
  3. a command line ***flag***:
     - streamlit run your_script.py --server.port 80

https://docs.streamlit.io/library/advanced-features/configuration

Telemetry

- Statistical information on the use by users is collected
- To disable telemetry, you must specify the configuration option

```
1 [browser]
2 gatherUsageStates = false
3
```

The server must be restarted to update configuration options

https://docs.streamlit.io/library/advanced-features/configuration

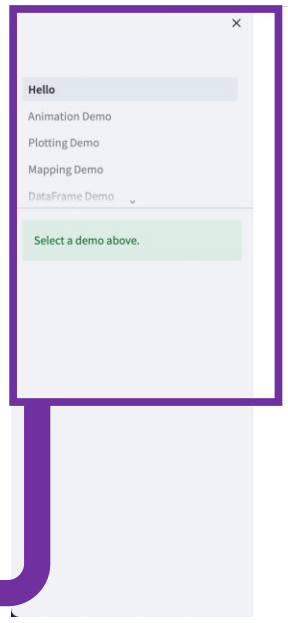
Start

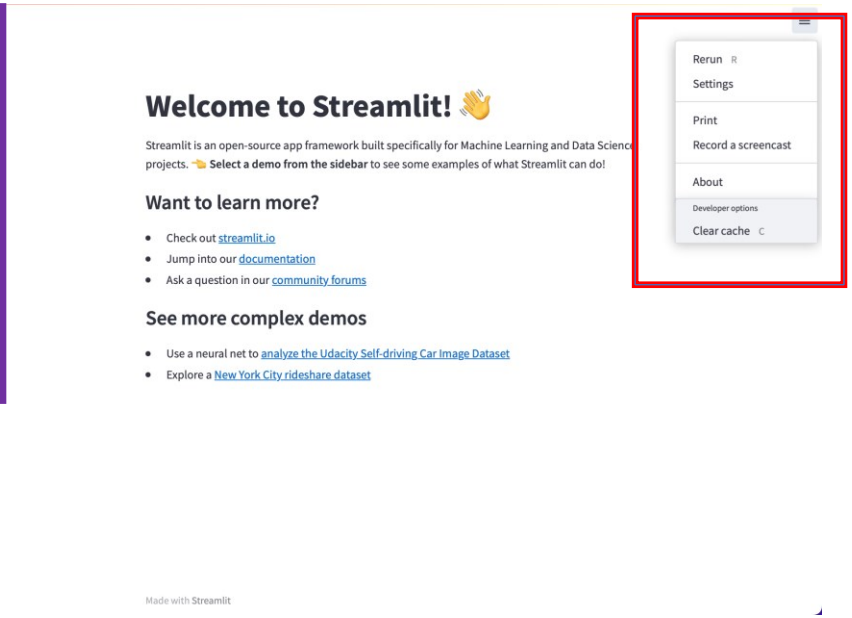
```
(streamlitTutorial) (base) ➜  streamlitTutorial streamlit hello

  Welcome to Streamlit. Check out our demo in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.89:8501

  Ready to create your own Python apps super quickly?
  Head over to https://docs.streamlit.io

  May you create awesome apps!
```

Command to start Streamlit

URL to reach the web server at port 8501

Hamburger menu



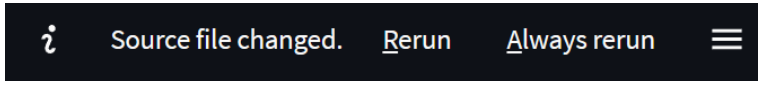


Sidebar with access to sample demos

Development

- Every time the Python script is saved, the application updates with a click on ***Rerun***, without the need to restart the server
- By choosing ***Always rerun***, the application updates automatically with each save, allowing you to immediately see the changes
- Whenever something needs to be updated on the screen (including user interactions), Streamlit launches the *top-to-bottom* script entirely.
- The server can be stopped with

  Ctrl+C



Project structure

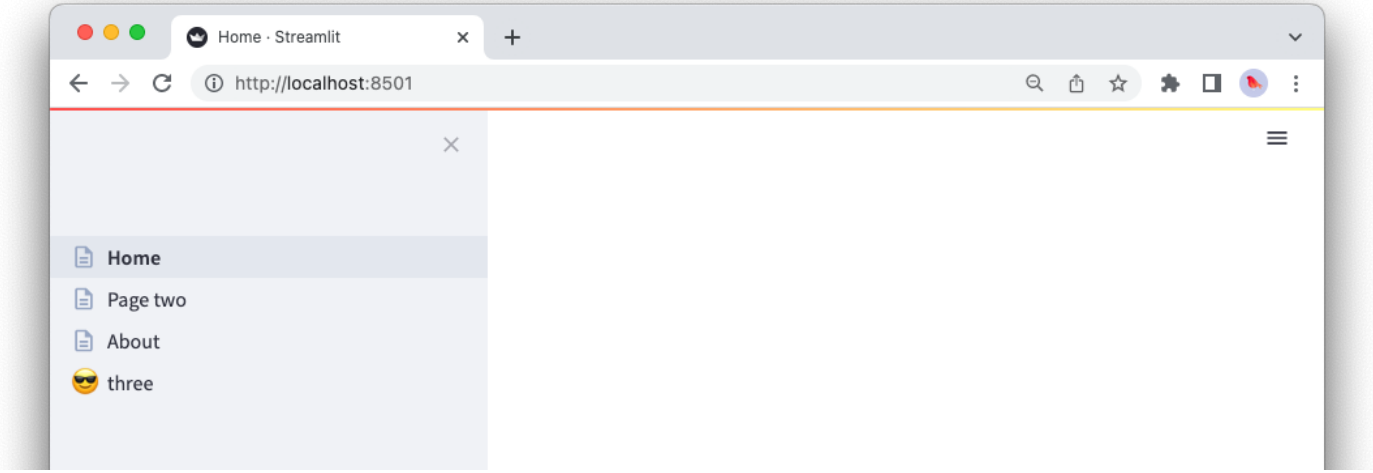
- Before you develop your app, it's important to define the project *directory* structure
- You need to define an ***entrypoint file*** that represents the main page to show to the user
- Other additional pages should be placed in a sub-folder ***pages***
- Pages globally share the same Python modules

```
Home.py # This is the file you run with "streamlit run"
└── pages/
    └── About.py # This is a page
    └── 2_Page_two.py # This is another page
    └── 3_😎_three.py # So is this
```

```python
# Home.py
import streamlit as st
```

Application pages

- Pages are defined by files *.py* within the *"pages/"* folder
- File names are transformed into page names
- The order is given by the number preceding the title and/or by the alphabetical order of the title itself.
- The number used as a prefix in the file name is not interpreted as part of the title



Page configuration

- Set the default page configuration
  - st.set_page_config(page_title=None, page_icon=None, layout="centered", initial_sidebar_state="auto", menu_items=None)

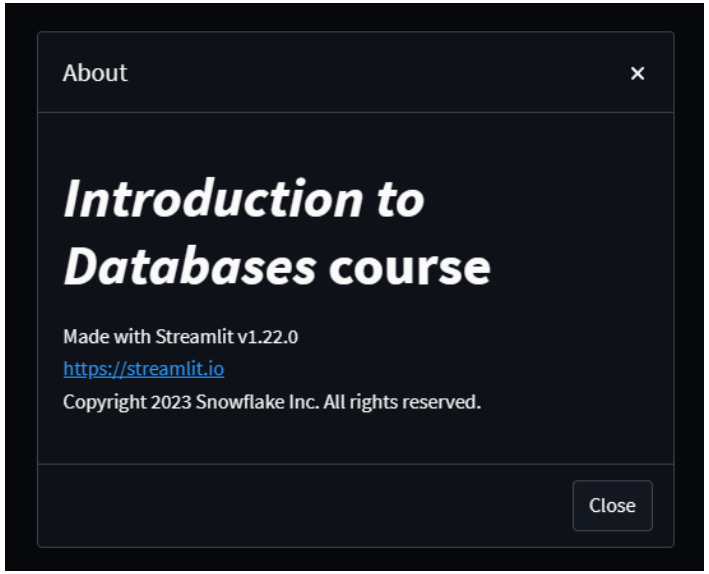
```python
import streamlit as st

st.set_page_config(
    page_title="My App",
    layout="wide",
    initial_sidebar_state="expanded"
)
```

It must be the first Streamlit command and set only once!

Customization of the hamburger menu

- Using the *menu_items* parameter, you can customize the items to be shown in the hamburger menu
- It must be formatted according to a dictionary in which the key is the element you want to change

```python
import streamlit as st

st.set_page_config(
    page_title="My App",
    layout="wide",
    initial_sidebar_state="expanded",
    menu_items={'Get Help': 'https://dbdmg.polito.it/',
                'Report a bug': "https://dbdmg.polito.it/",
                'About': "# *Introduction to Databases* course"
                }
)
```



Elements of Streamlit

- Widgets and elements specific to different types of activities and inputs
  - quickly integrate different features into your application
  - available through official documentation: https://docs.streamlit.io/library/api-reference
- Most significant categories:
  - Text elements
  - Input widgets
  - Layout
  - Visualization of data and graphs
  - Additional elements

Element arguments

- The various elements can be integrated without special configurations
  - Personalization via certain arguments
- Some arguments are common to all (or most) of the elements:
  - ***label***: describes to the user the functionality of the element (e.g. the name of a clickable button)
  - ***label_visibility*****:** determine label visibility (i.e., ”visible”, ”hidden”, ”collapsed”); the label should always be defined
  - ***disabled***: boolean flag to disable an element. Useful for making a widget available only if a certain condition occurs
  - ***use_container_width***: boolean flag to fit the size of the widget to that of the container it is part of
  - ***key:*** string or number to uniquely identify the widget. If omitted, it is generated based on content

Different items cannot have the same key!

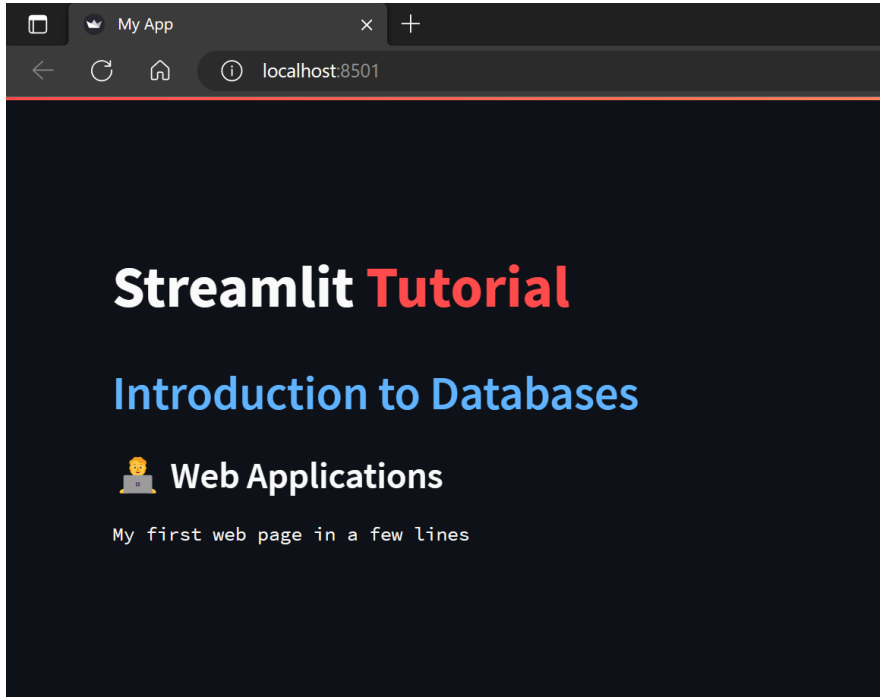
Text elements

Web Applications

Text elements

- Different ready-to-use text elements, with the ability to customize the color and insert *emojis*:
  - Title
  - Header
  - Sub-header
  - Text

```python
import streamlit as st

st.set_page_config(
    page_title="My App",
    layout="wide",
    initial_sidebar_state="expanded"
)
st.title("Streamlit :red[Tutorial]")
st.header(":blue[Introduction to Databases]")
st.subheader("👨‍💻 Web Applications")
st.text("My first web page in a few lines")
```



Markdown

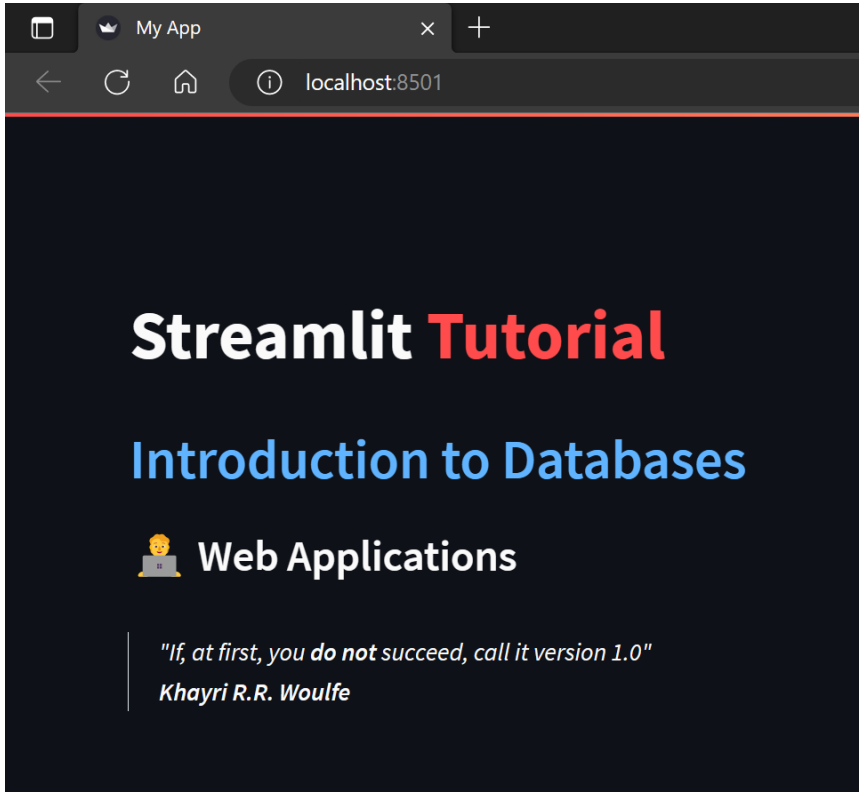
- It is possible to insert strings formatted according to the ***markdown*** language
- Markdown is used to format text quickly and easily, being more readable than other markup languages
- The most common syntax *(N.B. spaces are sometimes necessary!)*:

# Header 1	**bold**
## Header 2	> blockquote
### Header 3	* Item 1 * Item 2
italics	Line ␣ ␣ Break

https://www.markdownguide.org/basic-syntax/

Markdown example

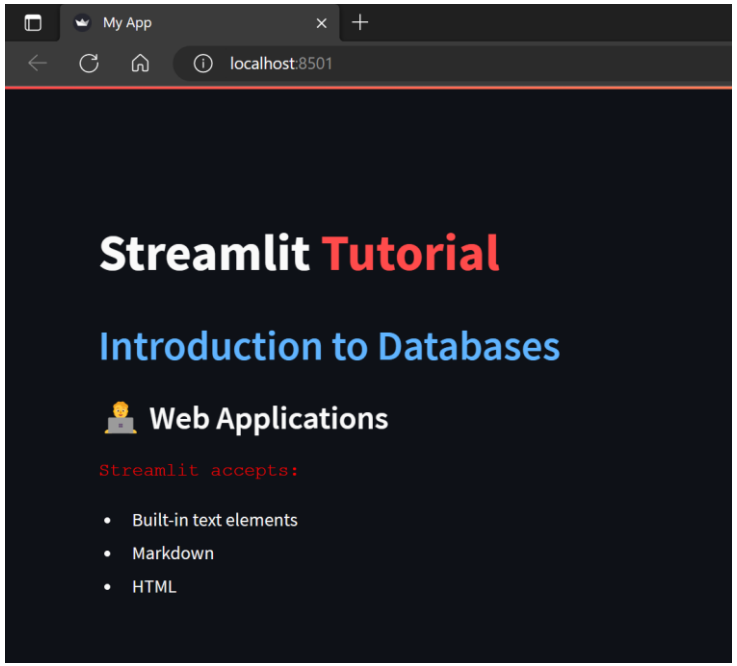
```python
import streamlit as st

st.set_page_config(
    page_title="My App",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.markdown("# Streamlit :red[Tutorial]")
st.markdown("## :blue[Introduction to Databases]")
st.markdown("### 👨‍💻 Web Applications")
st.markdown(""">_"If, at first, you **do not** succeed, call it version 1.0"
        **Khayri R.R. Woulfe**_""")
```



Markdown and HTML

- You can also use markdown to insert HTML code
- Useful for special customizations
- It is necessary to enable the use of HTML code
  - the feature is disabled by default to prevent the developer from inserting unsafe code

```python
import streamlit as st

st.set_page_config(
    page_title="My App",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.markdown("# Streamlit :red[Tutorial]")
st.markdown("## :blue[Introduction to Databases]")
st.markdown("### 🤦💻 Web Applications")
html_string="""<p style="font-family:courier;color:red">Streamlit accepts:</p>
        <ul>
            <li> Built-in text elements </li>
            <li> Markdown </li>
            <li> HTML </li>
        </ul>"""
st.markdown(html_string, unsafe_allow_html=True)
```



Write

- Allows to write in the app the arguments that are passed to it
  - st.write(*args, unsafe_allow_html=False, **kwargs)
- Universal and flexible widget that behaves differently based on the passed argument
  - accepts different types of arguments and renders them accordingly
  - multiple arguments can be passed and will be represented
  - allows to represent different Python objects (e.g., figure, dataframe, dictionaries, errors, functions and modules) also in interactive mode

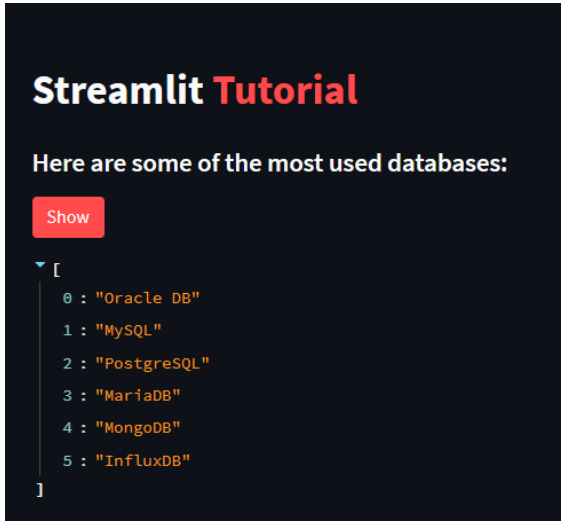
Input Widgets

Web Applications

Button

- Allows you to show a simple button that can be clicked by the user
  - st.button(**label**, key=None, help=None, on_click=None, args=None, kwargs=None, **type="secondary"**, disabled=False, use_container_width=False)

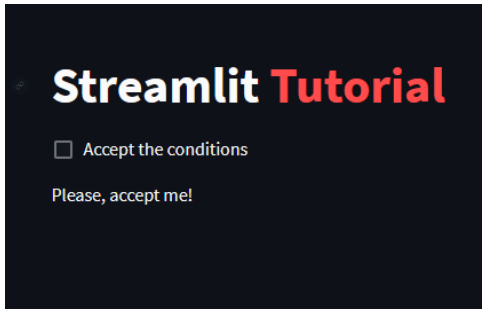
```
import streamlit as st

st.markdown("# Streamlit :red[Tutorial]")
st.markdown("#### Here are some of the most used databases:")

db_list=["Oracle DB", "MySQL", "PostgreSQL", "MariaDB", "MongoDB", "InfluxDB"]

if st.button("Show", type="primary"):
    st.write(db_list)
```

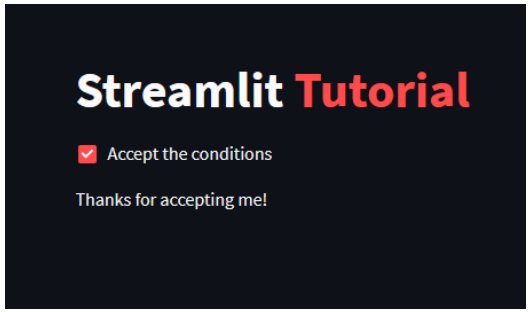


Checkbox

- Allows you to show a ***checkbox*** to check and perform a follow-up action
  - st.checkbox(label, value=False, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")
- Returns True or False based on checkbox status

```python
import streamlit as st

st.markdown("# Streamlit :red[Tutorial]")

if st.checkbox("Accept the conditions"):
    st.write("Thanks for accepting me!")
else:
    st.write("Please, accept me!")
```

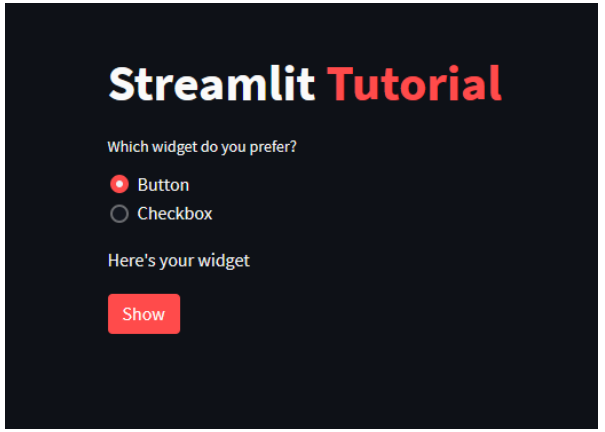


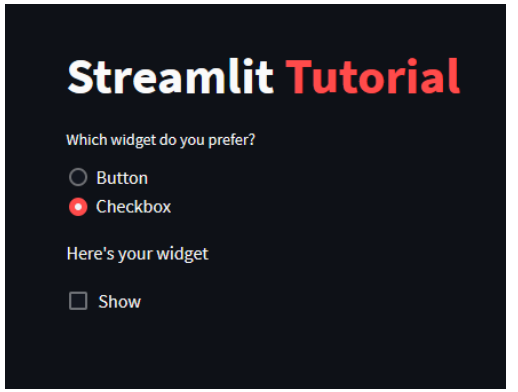


Radio Button

- Allows you to insert a ***radio button*** with which the user can make an exclusive choice among the proposed alternatives
  - st.radio(label, options, index=0, format_func=special_internal_function, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, horizontal=False, label_visibility="visible")
- Returns the chosen option

```python
import streamlit as st

st.markdown("# Streamlit :red[Tutorial]")
widget = st.radio("Which widget do you prefer?", ["Button", "Checkbox"])

st.write("Here's your widget")

if widget == "Button":
    if st.button("Show", type="primary"):
        # Perform what you want
        pass
else:
    if st.checkbox("Show"):
        # Perform what you want
        pass
```

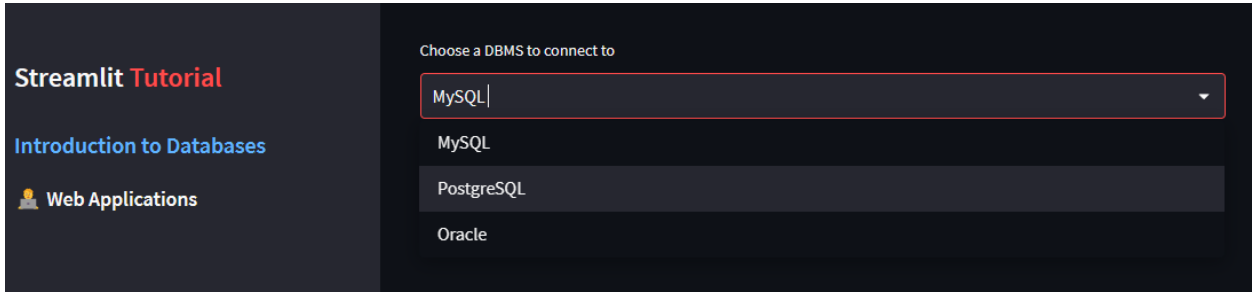




Select Box

- Allows you to insert a drop-down ***selection box*** with which the user can choose between the various alternatives
  - st.selectbox(label, options, index=0, format_func=special_internal_function, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")
- Returns the chosen option

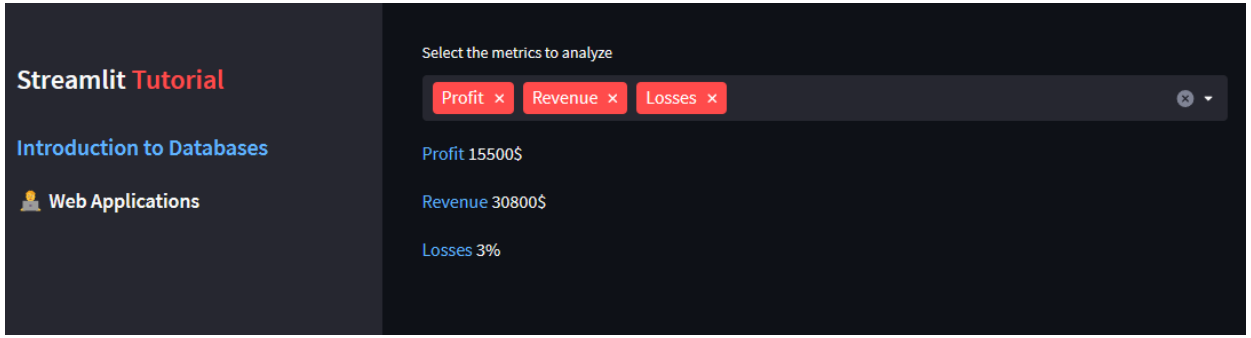
```
import streamlit as st

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("🧑‍💻 Web Applications")

option = st.selectbox(
    'Choose a DBMS to connect to',
    ('MySQL', 'PostgreSQL', 'Oracle'))
```



Multiselect

- Allows the user to choose multiple alternatives among those proposed
  - st.multiselect(label, options, default=None, format_func=special_internal_function, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible", max_selections=None)
- The *default* parameter specifies the list of options selected at startup
- The *max_selections* parameter defines the maximum number of options that can be selected
- Returns the list of selected options

```
import streamlit as st

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("🧑‍💻 Web Applications")

my_dict = {"Profit": "15500$", "Revenue": "30800$", "Losses": "3%"}

options = st.multiselect(
    'Select the metrics to analyze',
    ['Profit', 'Revenue', 'Losses'])

for option in options:
    st.write(f':blue[{option}] {my_dict[option]}')
```



Slider

- Offers a ***slider*** that accepts: int, float, time, date and datetime
  - st.slider(label, min_value=None, max_value=None, value=None, step=None, format=None, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")
- Allows you to select both a single value and a range of values
- Returns the selected value or tuple (for ranges)
- The *min_value* (*default* 0 if int, 0.0 if float) and *max_value* (*default* 100 if int, 1.0 if float) parameters define the minimum and maximum allowed value, respectively.

Slider

- Offers a ***slider*** that accepts: int, float, time, date and datetime
  - st.slider(label, min_value=None, max_value=None, value=None, step=None, format=None, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")
- The *value* parameter defines the assumed value when the slider is loaded for the first time
  - If set with a tuple, create a slider with the selectable range
  - by default it is set to *min_value*
- The *step parameter* defines the interval between one value and another (*default* 1 if int, 0.01 if float)

Slider Example

```python
import streamlit as st
from datetime import datetime, time

# simple slider
age = st.slider('How old are you?', 0, 130, 25)
st.write("You are", age, "years old")

# range slider
values = st.slider('Selecte a range of values',
                   0.0, 100.0, (25.0, 75.0))
st.write('Values:', values)

# range time slider
appointment = st.slider(
    "Book your appointment:",
    value=(time(11, 30), time(12, 45)))
st.write(f"You have booked for:", appointment[0], '-', appointment[1])

# datetime slider
start_time = st.slider(
    "When you want to start?",
    value=datetime(2020, 1, 1, 9, 30),
    format="MM/DD/YY - hh:mm")
st.write("Start:", start_time)
```

Text and Number

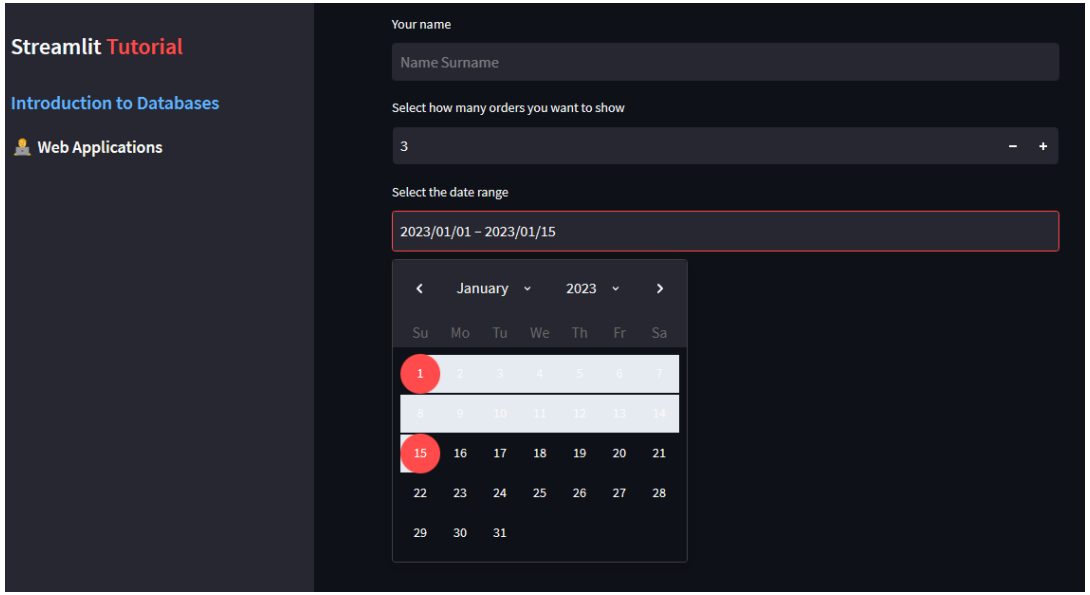
- ***Text input*** offers the possibility of single-line text input
  - st.text_input(label, value="", max_chars=None, key=None, type="default", help=None, autocomplete=None, on_change=None, args=None, kwargs=None, *, placeholder=None, disabled=False, label_visibility="visible")

- The ***number input*** allows you to pass a number that can be written from the keyboard or using the '+' and '-' keys
  - st.number_input(label, min_value=None, max_value=None, value=, step=None, format=None, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")

Date input

- Provides an ideal widget for selecting a ***date*** on a calendar
  - st.date_input(label, value=None, min_value=None, max_value=None, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")
- The *value* parameter also accepts a list/tuple to enable a date range

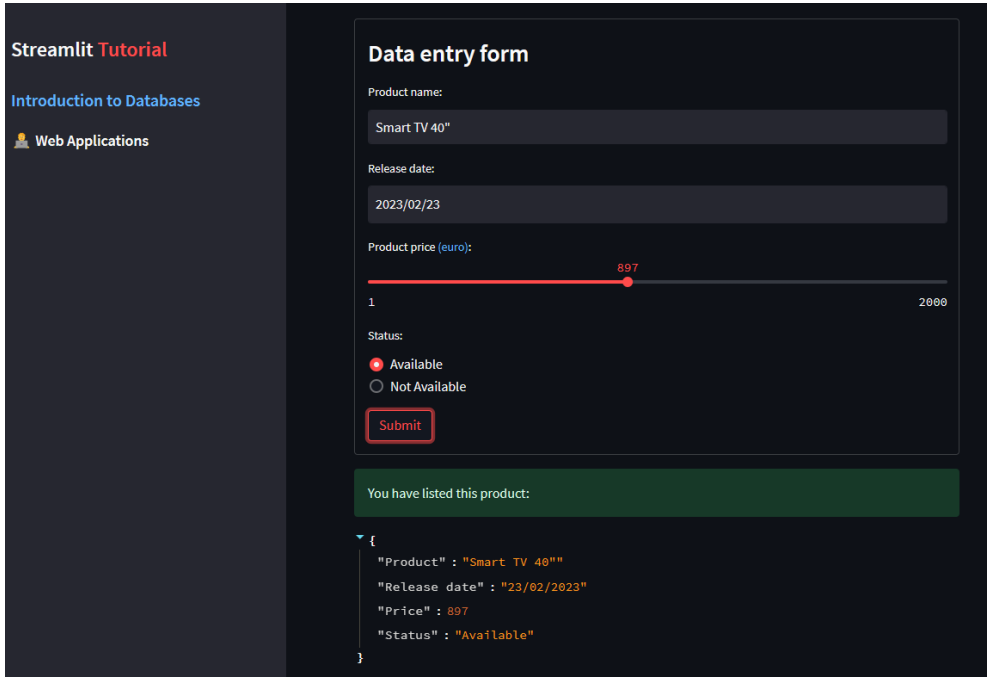
```python
import streamlit as st
import datetime

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("🧑‍💻 Web Applications")

name = st.text_input('Your name', placeholder="Name Surname")
number = st.number_input('Select how many orders you want to show', min_value=1, value=3)
date = st.date_input(
    "Select the date range",
    value=(datetime.date(2023, 1, 1), datetime.date(2023, 1, 15)))
```



Form

- Allows you to group several elements on a ***form*** (container)
  - ➢ st.form(key, clear_on_submit=False)
- It has integrated a *Submit* button that collects all the values acquired by the different widgets
- This parameter *clear_on_submit* if True resets widget values after user clicks the Submit button

```python
import streamlit as st
import datetime

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("Web Applications")

with st.form("form"):
    st.subheader("Data entry form")
    product = st.text_input("Product name:")
    data = st.date_input("Release date:", value=datetime.datetime(2023, 1, 1))

    price = st.slider("Product price :blue[(euro)]:", 1, 2000)
    status = st.radio("Status:", ("Available", "Not Available"))

    # Every form must have a submit button
    submitted = st.form_submit_button("Submit")

if submitted:
    st.success("You have listed this product:")
    st.write({"Product": product, "Release date": data.strftime('%d/%m/%Y'), "Price": price, "Status": status})
```



Data visualization

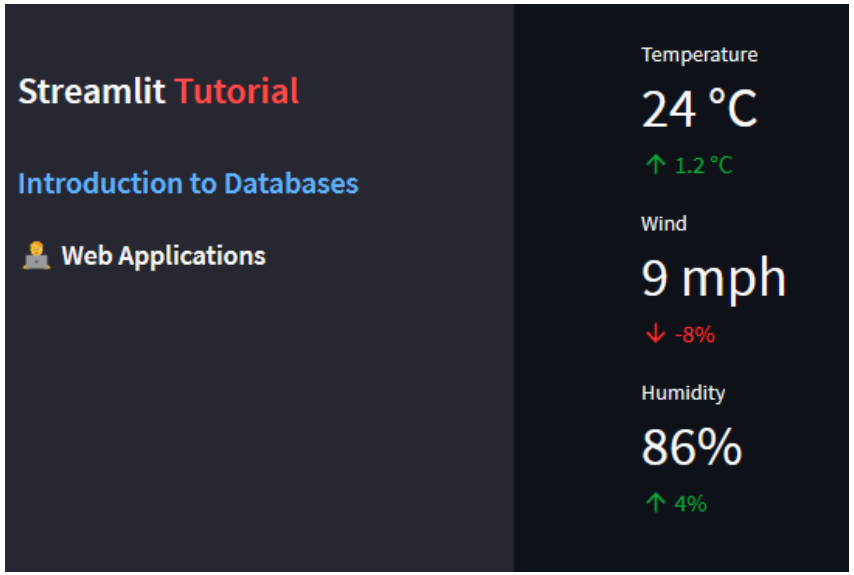
Web Applications

Metrics

- Displays a ***metric*** with a specific font, giving you the option to add a variation indicator
  - st.metric(label, value, delta=None, delta_color="normal", help=None, label_visibility="visible")

- The *delta* parameter indicates the variation

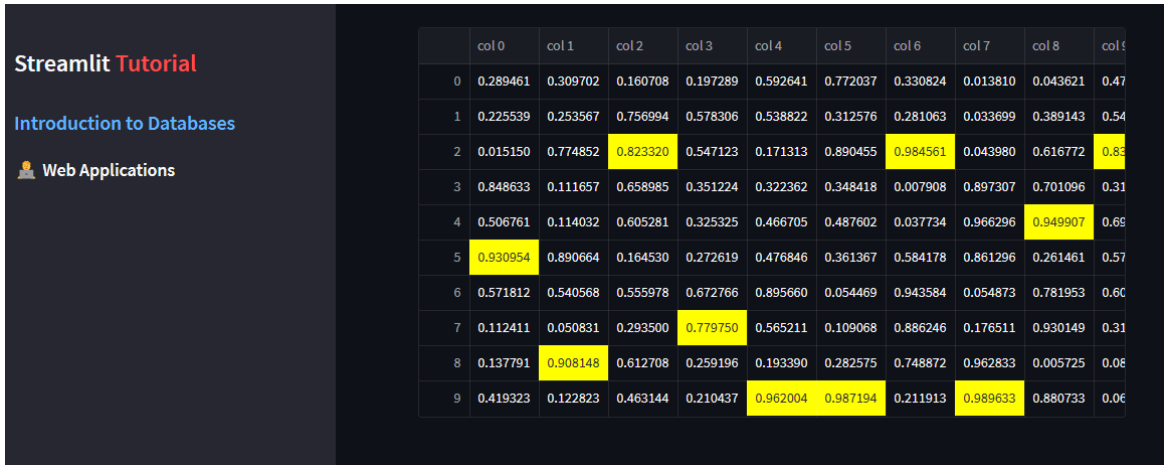
```
import streamlit as st

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("👨‍💻 Web Applications")

st.metric("Temperature", "24 °C", "1.2 °C")
st.metric("Wind", "9 mph", "-8%")
st.metric("Humidity", "86%", "4%")
```



Dataframe

- Displays pandas ***dataframes*** in the form of interactive tables
  - st.dataframe(data=None, width=None, height=None, *, use_container_width=False)

```python
import streamlit as st
import numpy as np
import pandas as pd

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("🧑‍💻 Web Applications")

df = pd.DataFrame(
    np.random.rand(10, 20),
    columns=('col %d' % i for i in range(20)))

st.dataframe(df.style.highlight_max(axis=0))
```



	col 0	col 1	col 2	col 3	col 4	col 5	col 6	col 7	col 8	col 9
0	0.289461	0.309702	0.160708	0.197289	0.592641	0.772037	0.330824	0.013810	0.043621	0.47
1	0.225539	0.253567	0.756994	0.578306	0.538822	0.312576	0.281063	0.033699	0.389143	0.54
2	0.015150	0.774852	0.823320	0.547123	0.171313	0.890455	0.984561	0.043980	0.616772	0.83
3	0.848633	0.111657	0.658985	0.351224	0.322362	0.348418	0.007908	0.897307	0.701096	0.31
4	0.506761	0.114032	0.605281	0.325325	0.466705	0.487602	0.037734	0.966296	0.949907	0.69
5	0.930954	0.890664	0.164530	0.272619	0.476846	0.361367	0.584178	0.861296	0.261461	0.57
6	0.571812	0.540568	0.555978	0.672766	0.895660	0.054469	0.943584	0.054873	0.781953	0.60
7	0.112411	0.050831	0.293500	0.779750	0.565211	0.109068	0.886246	0.176511	0.930149	0.31
8	0.137791	0.908148	0.612708	0.259196	0.193390	0.282575	0.748872	0.962833	0.005725	0.08
9	0.419323	0.122823	0.463144	0.210437	0.962004	0.987194	0.211913	0.989633	0.880733	0.06

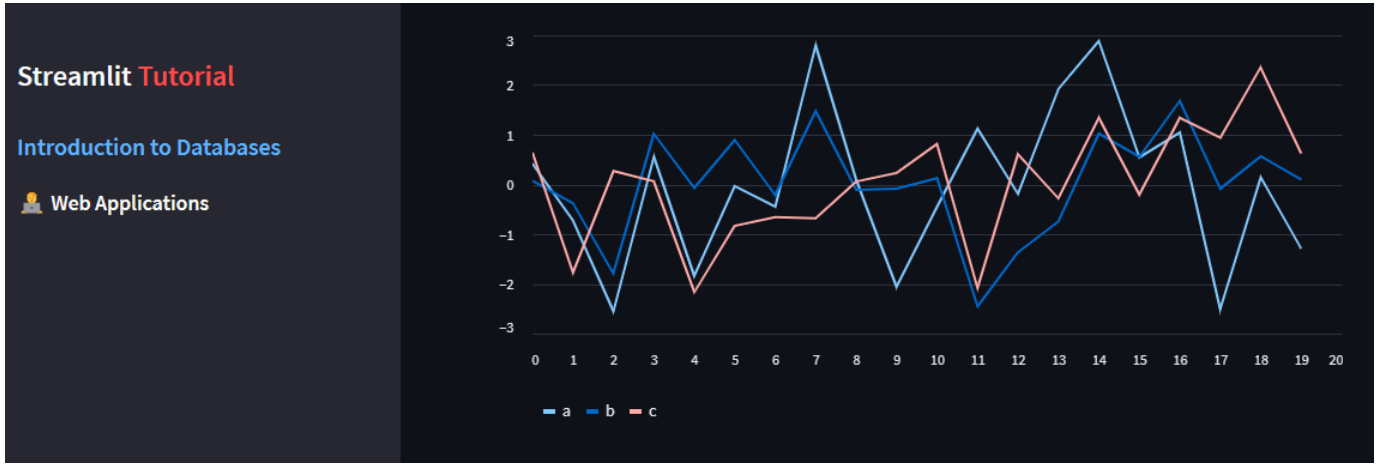
Charts

- Several libraries are supported for the graphical representation of data through ***interactive charts***
  - Matplotlib
  - Plotly
  - Altair
  - deck.gl (maps and 3D graphs)
- To speed up the integration of the most common charts, some are natively integrated into Streamlit (with less customization):
  - Line chart
  - Area chart
  - Bar chart
  - Scatterplot on map

Line Chart

- It allows you to represent a ***line chart*** and is based on Altair
  - st.line_chart(data=None, *, x=None, y=None, width=0, height=0, use_container_width=True)
- Ideal for simple plots to include quickly and easily
- *x* and *y* specify the name of the columns to use on the axes
- *Width* and *height* parameters specify dimensions in pixels

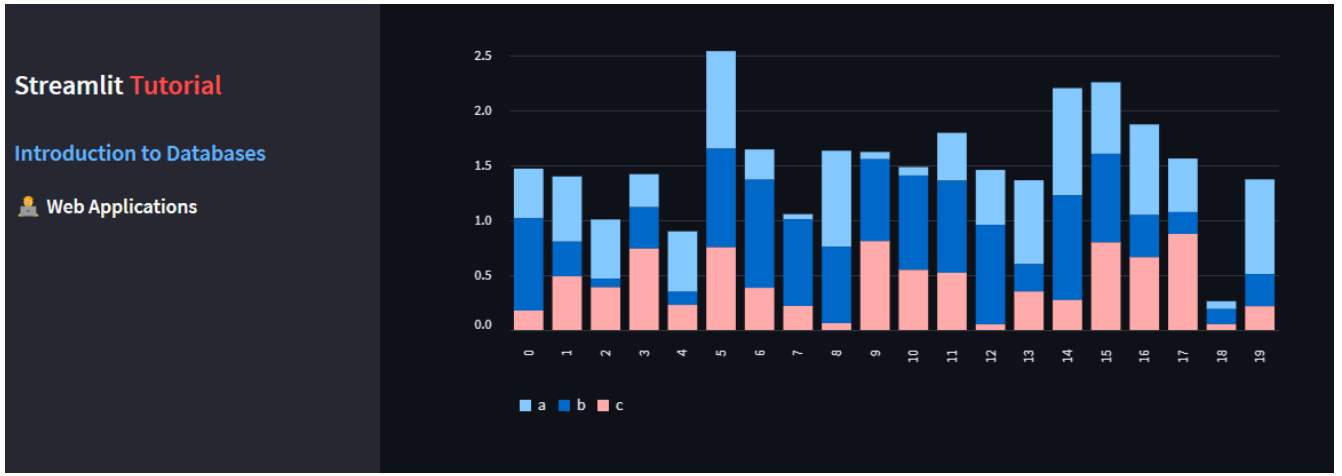
```
import streamlit as st
import numpy as np
import pandas as pd

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("👨‍💻 Web Applications")

chart_data = pd.DataFrame(
    np.random.randn(20, 3),
    columns=['a', 'b', 'c'])

st.line_chart(chart_data)
```



Bar Chart

- It allows you to represent a ***bar chart*** and is based on Altair
  - st.bar_chart(data=None, *, x=None, y=None, width=0, height=0, use_container_width=True)
- Ideal for simple plots to include quickly and easily
- *x* and *y* specify the name of the columns to use on the axes
- *Width* and *height* parameters specify dimensions in pixels

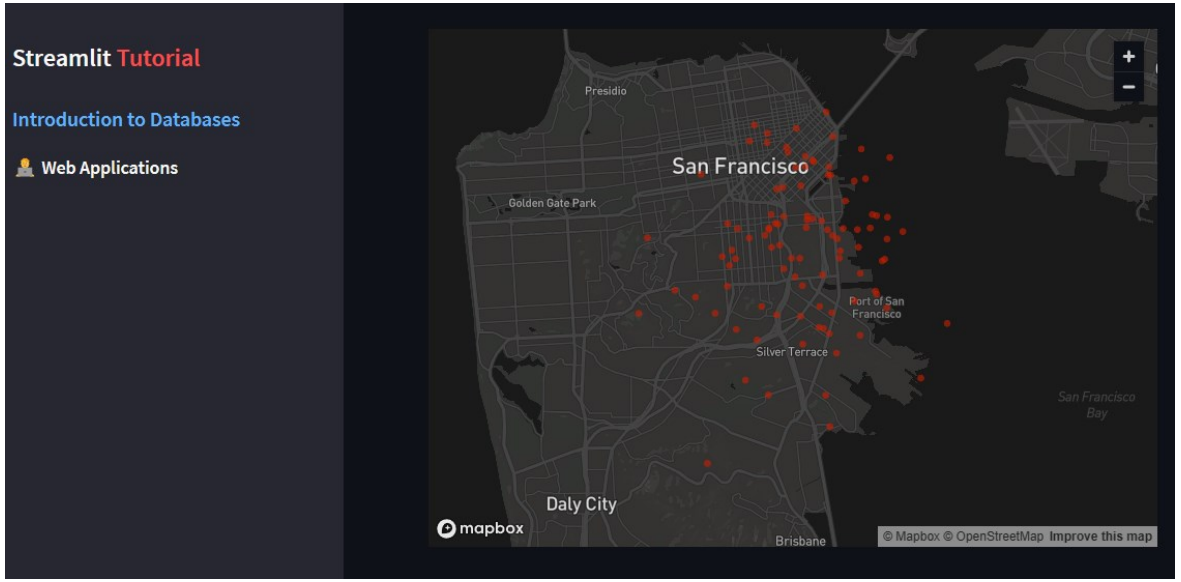
```python
import streamlit as st
import numpy as np
import pandas as pd

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("👨‍💻 Web Applications")

chart_data = pd.DataFrame(
    np.random.rand(20, 3),
    columns=['a', 'b', 'c'])

st.bar_chart(chart_data)
```



Map

- Allows you to visualize ***points on map*** and it is based on Pydeck
  - st.map(data=None, zoom=None, use_container_width=True)
- The *data* parameter must have two columns:
  1. Latitude called 'lat', *'latitude', 'LAT', 'LATITUDE'*
  2. Longitude called 'lon', *'longitude', 'LON', 'LONGITUDE'*
- The map relies on the external service Mapbox and requires a token (currently offered automatically by Streamlit)

```python
import streamlit as st
import numpy as np
import pandas as pd

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("🧑‍💻 Web Applications")

df = pd.DataFrame(
    np.random.randn(100, 2) / [50, 50] + [37.76, -122.4],
    columns=['lat', 'lon'])

st.map(df)
```



Additional elements

Web Applications

Additional elements

- Session state
- Elements to customize the application
- For example:
  - status messages
  - progress bar
  - spinner

Session state

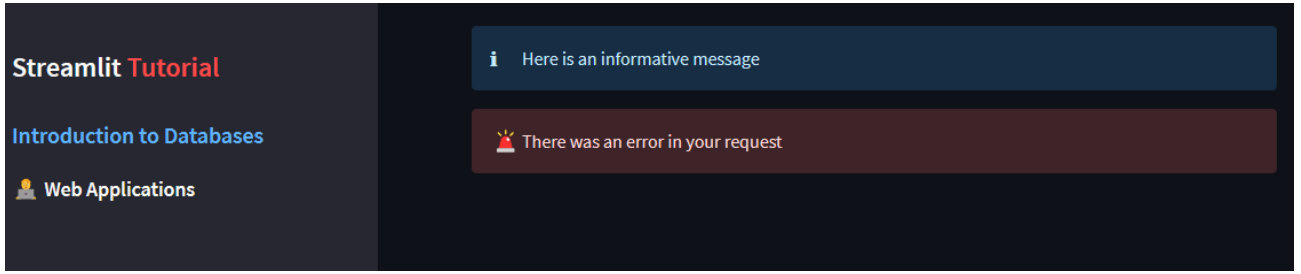
- A way to share variables between runs and pages, similar to a Python dictionary
  - st.session_state
- You must initialize the variable before trying to access it or an exception is raised
- Each widget with a *key* is automatically added to the Session State

```python
import streamlit as st

# Initialization
if 'key' not in st.session_state:
    st.session_state['key'] = 'value'

# Assign the value
if 'key' not in st.session_state:
    st.session_state.key = 'value'
```

Status messages

- Status messages are useful for rendering warnings, errors, or success messages
  - st.error(body, *, icon=None)
  - st.warning(body, *, icon=None)
  - st.info(body, *, icon=None)
  - st.success(body, *, icon=None)

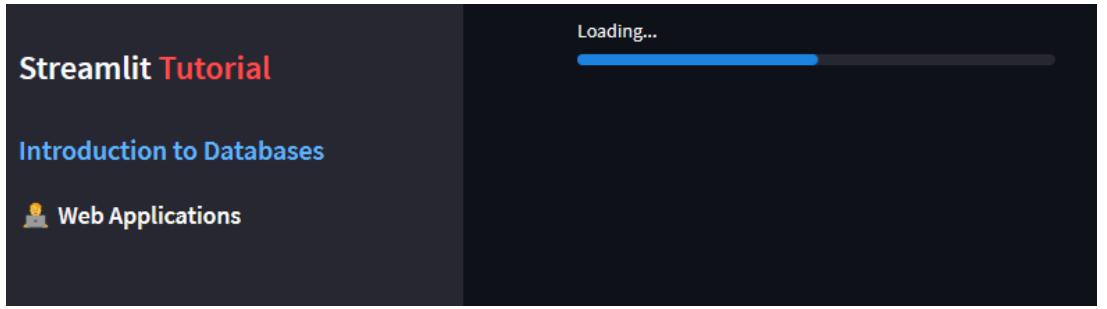
```
import streamlit as st

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("👨‍💻 Web Applications")

st.info('Here is an informative message', icon="ℹ️")
st.error("There was an error in your request", icon="🚨")
```



Progress bar

- To visualize the progress you can use a ***progress bar***
  - st.progress(value, text=None)
- The *value* parameter is 0 to 100 for integers, 0.0 to 1.0 for float
- The *text* parameter is the text to be shown above the progress bar

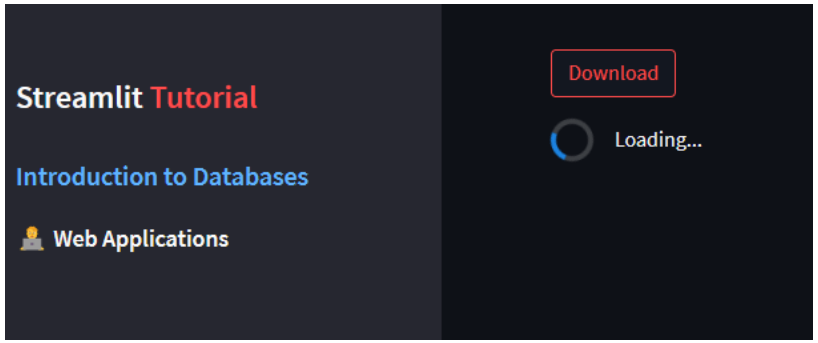
```python
import streamlit as st
import time

col1, col2 = st.columns(2)
st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("👨‍💻 Web Applications")

progress_text = "Loading..."
my_bar = col1.progress(0, text=progress_text)

for percent_complete in range(100):
    time.sleep(0.1)
    my_bar.progress(percent_complete + 1, text=progress_text)
```



Spinner

- To show temporary text while a block of code is executed, you can use a ***spinner*** to show to the user
  - st.spinner(text="In progress...")
- The *text* parameter is the message to be shown with the spinner until execution finishes

```
import streamlit as st
import time

col1, col2 = st.columns(2)
st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("👨‍💻 Web Applications")

if col1.button("Download"):
    with st.spinner("Loading..."):
        time.sleep(3)
    col2.success('Done!')
```



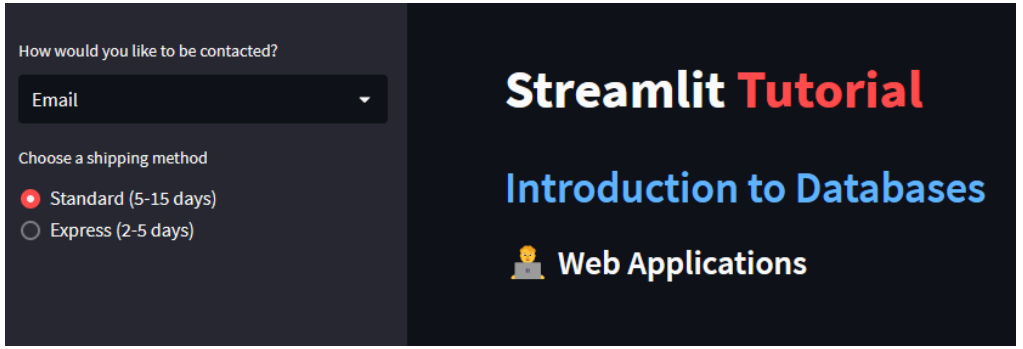
Layout

Web Applications

Sidebar

- The ***sidebar*** is very useful for adding elements on the left, leaving the user full concentration on the main application
- Accept both ***object notation*** and ***with notation***
- Layout elements can typically be used as streamlit objects and therefore contain several elements

```python
import streamlit as st

st.set_page_config(
    page_title="My App",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.title("Streamlit :red[Tutorial]")
st.header(":blue[Introduction to Databases]")
st.subheader("👨‍💻 Web Applications")

# Using object notation
add_selectbox = st.sidebar.selectbox(
    "How would you like to be contacted?",
    ("Email", "Home phone", "Mobile phone")
)

# Using "with" notation
with st.sidebar:
    add_radio = st.radio(
        "Choose a shipping method",
        ("Standard (5-15 days)", "Express (2-5 days)")
    )
```

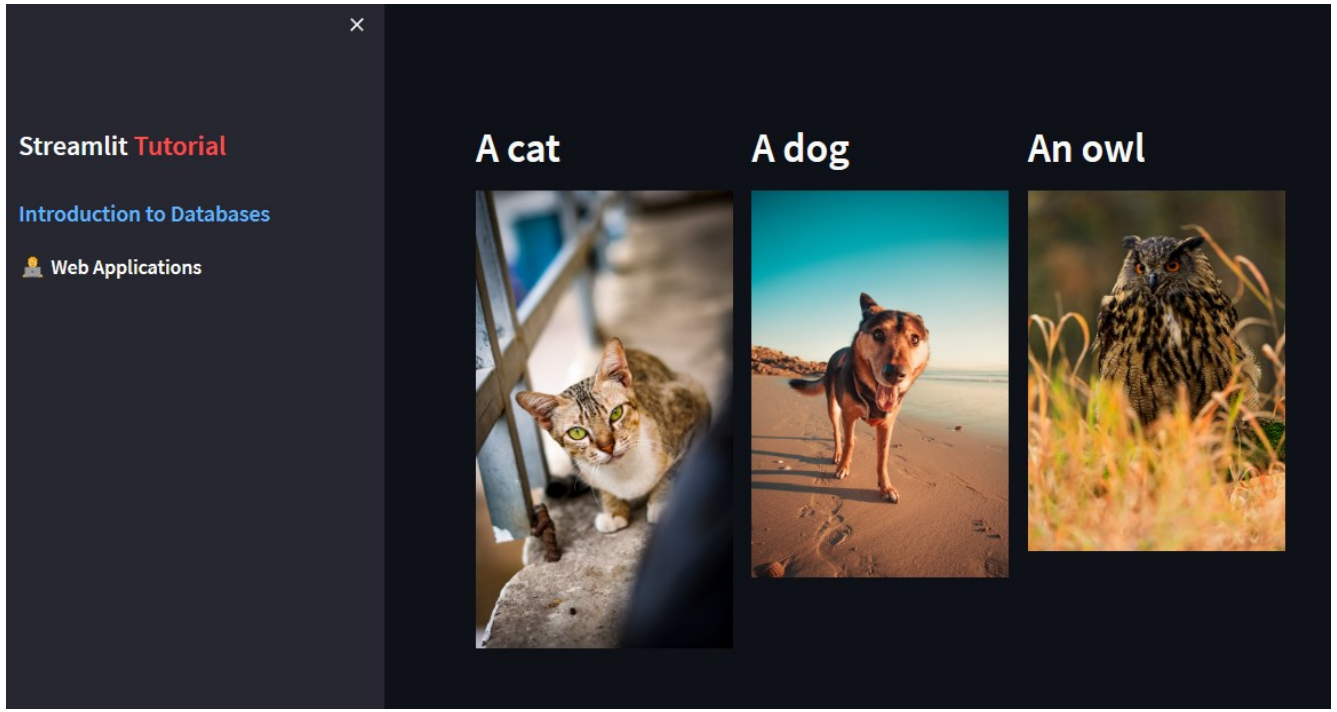


Columns

- To divide space into side-by-side containers, you can divide the page into ***columns***
  - st.columns(spec, *, gap="small")
- The *spec* parameter can be an integer or a list of numbers
  - if an integer, indicates the number of columns to be inserted, all with the same width
  - If a list, a column is created for each number with width proportional to the specified number
- The *gap* parameter indicates the distance between the columns
- The list of containers (i.e., columns) is returned
- Accept both ***object notation*** and ***with notation***

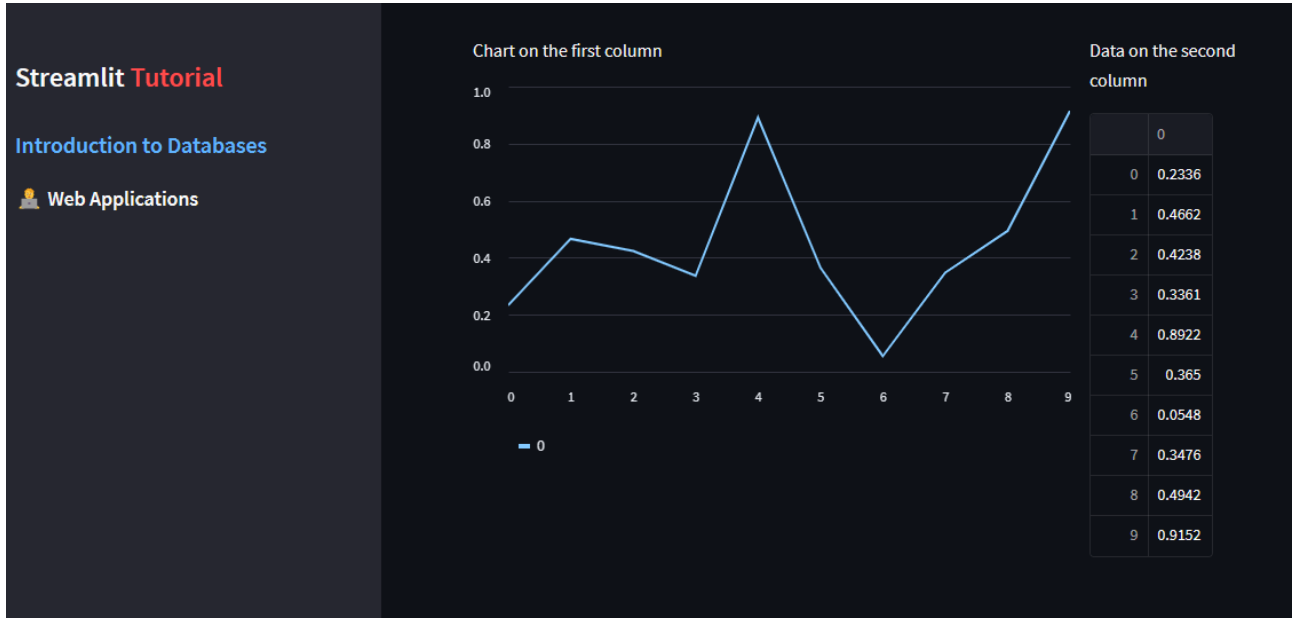
Columns: with notation example

```python
import streamlit as st

st.set_page_config(
    page_title="My App",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("🧠💻 Web Applications")

col1, col2, col3 = st.columns(3)

with col1:
    st.header("A cat")
    st.image("https://static.streamlit.io/examples/cat.jpg")

with col2:
    st.header("A dog")
    st.image("https://static.streamlit.io/examples/dog.jpg")

with col3:
    st.header("An owl")
    st.image("https://static.streamlit.io/examples/owl.jpg")
```



Columns: object notation example

```python
import streamlit as st
import numpy as np

st.set_page_config(
    page_title="My App",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("🧑‍💻 Web Applications")

col1, col2 = st.columns([3, 1])
data = np.random.rand(10, 1)

col1.write("Chart on the first column")
col1.line_chart(data)

col2.write("Data on the second column")
col2.write(data)
```



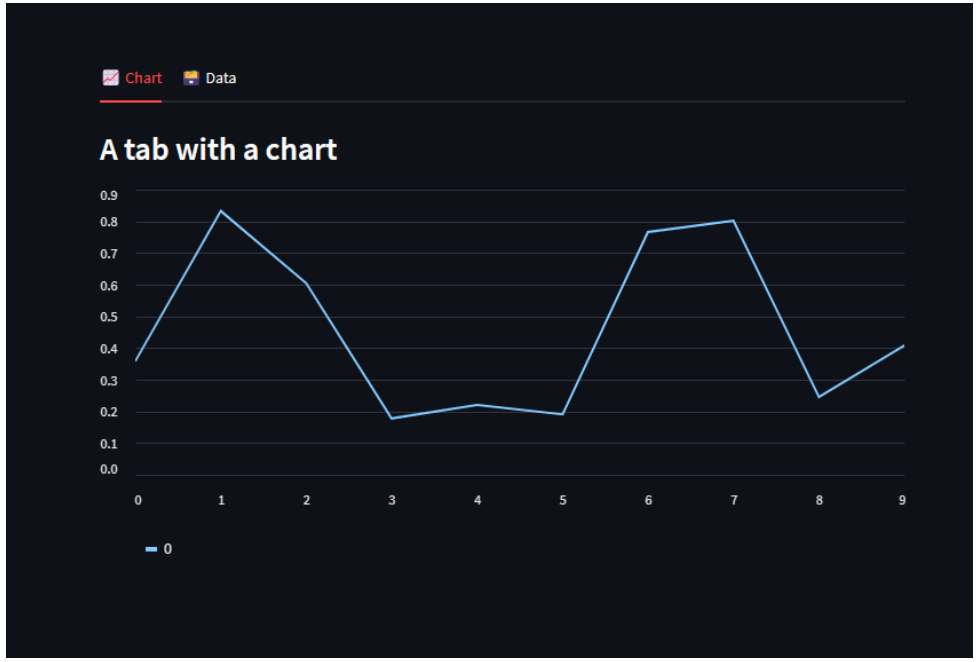
Tabs

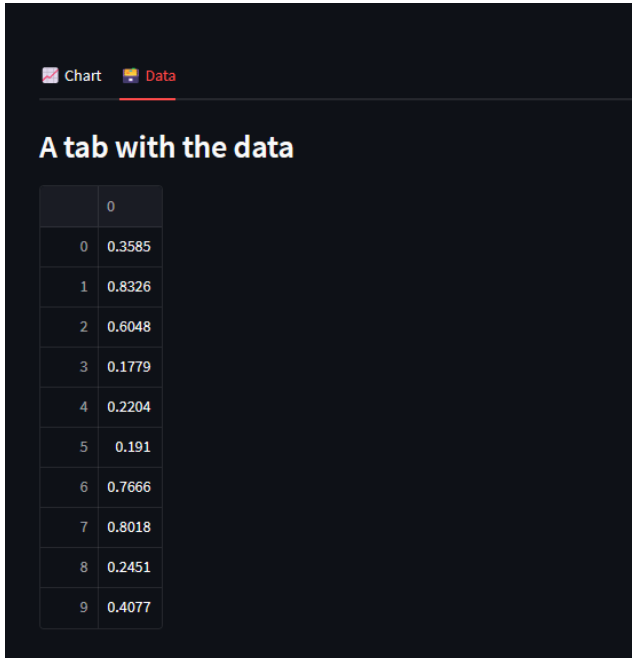
- Tabs allow a more structured organization of content
- The user can easily navigate between one tab and another
- To have separate containers, you can use the ***tabs***
  - st.tabs(tabs)

- The *tabs* parameter is a list of strings, each representing the name of a tab
- The first tab is the one selected by default
- As with columns, returns a list of containers
- Accepts the ***with notation***

Tabs example

```python
import streamlit as st
import numpy as np

tab1, tab2 = st.tabs(["📈 Chart", "🗃 Data"])
data = np.random.rand(10, 1)

tab1.subheader("A tab with a chart")
tab1.line_chart(data)

tab2.subheader("A tab with the data")
tab2.write(data)
```



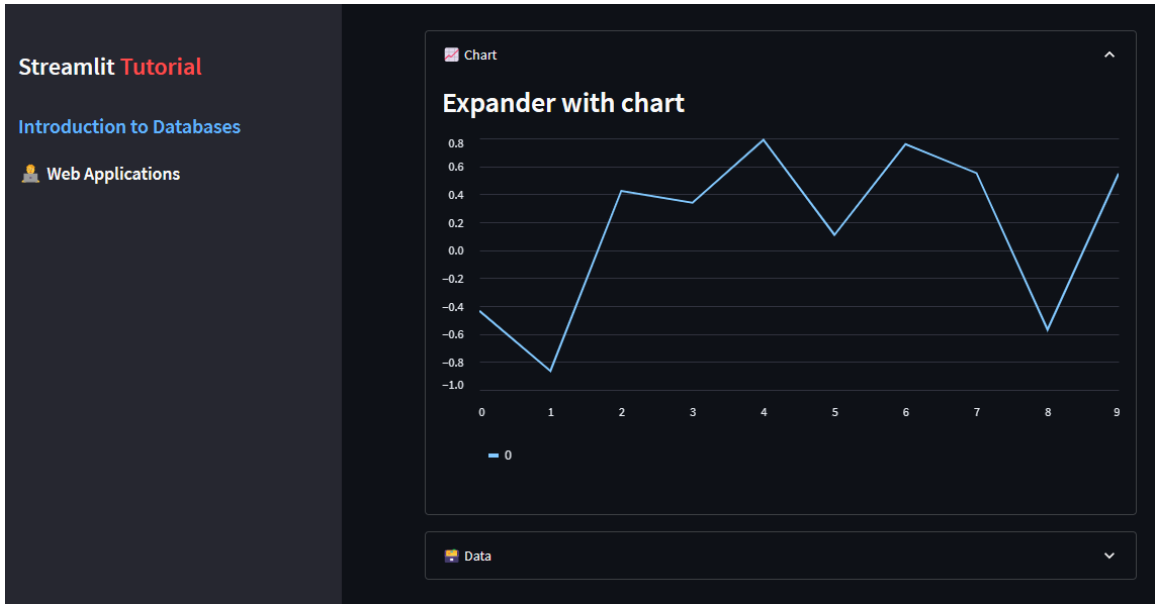


Expander

- ***Expanders*** allow you to define containers that the user can choose whether to open or close
  - ➢ st.expander(label, expanded=False)

- The *label* parameter represents the name of the expander
- The *expanded* parameter represents the default state of the expander, whether open or closed

```python
import streamlit as st
import numpy as np

st.sidebar.title("Streamlit :red[Tutorial]")
st.sidebar.header(":blue[Introduction to Databases]")
st.sidebar.subheader("Web Applications")

data = np.random.randn(10, 1)

with st.expander("Chart", expanded=True):
    st.subheader("Expander with chart")
    st.line_chart(data)

with st.expander("Data", expanded=False):
    st.subheader("Expander with data")
    st.write(data)
```



Editing Themes

- You can change the style and colors of the interface to create a custom application
- From the menu go to *Settings* and then to *Theme*
- You can choose between ***Light*** and ***Dark*** mode or change them by creating your own theme (changing the colors and font)
- In this way you can experiment live your customizations before copying them into the configuration file within the section [theme]